NASA Technical Paper 1426

NASF Transposition Network: A Computing Network for Unscrambling p-Ordered Vectors

Raymond S. Lim

APRIL 1979

NASA

NASA Technical Paper 1426

NASF Transposition Network: A Computing Network for Unscrambling p-Ordered Vectors

Raymond S. Lim
Ames Research Center
Moffett Field, California

National Aeronautics
and Space Administration

**Scientific and Technical
Information Office**

1979

BLank
Page

SYMBOLS

$a \mid b$	a divides b
$a \nmid b$	a does not divide b
g	an element of G
gcd	greatest common divisor
G	a group
i	a subscript to X_i, or just a general subscript
j	a subscript to M_j, or just a general subscript
k	a positive integer, or a generating element of the cyclic group M_N
L	a positive number, or the number of levels of k-apart interconnection network
m	a positive integer, or the number of routing requires in a k-apart interconnection network
mod N	modulo N
M_j	the jth memory module of an N-module array memory
M_N	a multiplicative cyclic group with $N - 1$ elements
n	number of elements in the vectors X
N	a positive integer, the number of memory modules in an array memory, or the number of registers in an array register
p	a positive integer, or the separation distance between the ith and the (i + 1)th elements of a p-ordered vector; p is also called the skip distance
P	a prime number
pq	two positive integers p and q to denote a pq-ordered vector; p is the skip distance and q is the separation distance
r	primitive root
R	number of processors
t	order of g
TN	transportation network

v	a positive integer equal to 2^i
X	a one-dimensional vector consists of n elements
X_i	the ith element of an n-element vector X
ε	contains in
$\lceil x \rceil$	smallest integer greater than or equal to x (ceiling of x)
$\lfloor x \rfloor$	smallest integer less than or equal to x (floor of x)

NASF TRANSPOSITION NETWORK:

A COMPUTING NETWORK FOR UNSCRAMBLING p-ORDERED VECTORS

Raymond S. Lim

Ames Research Center

SUMMARY

This paper presents a tutorial description of a transportation network (TN) proposed by the Burroughs Corporation for the Numerical Aerodynamic Simulation Facility (NASF). The description is presented from the viewpoints of design, programming, and application. The TN is a programmable combinational logic network that connects 521 memory modules to 512 processors, where gcd(521,512) = 1. The primary purpose of the TN is to transpose (or unscramble) p-ordered vectors to 1-ordered vectors in one cycle. For unscrambling pq-ordered vectors, the TN speed is degenerated to several cycles. The TN design, which is evolved from the Swanson network, is based upon the concept of cyclic groups from abstract algebra and primitive roots and indices from number theory. The design can be implemented by one level of barrel switch plus a fixed wiring pattern and its inverse. The connection of this fixed wiring pattern is from p to m according to $k^m \equiv p \pmod{N}$, where k is a primitive root of the prime N, m is the index of p relative to k, and p is an element of the cyclic group of order N - 1 generated by k. The programming of the TN is very simple, requiring only 20 bits: 10 bits for offset control and 10 bits for barrel switch shift control. This simple control is executed by the control unit (CU), not the processors. For this reason, any memory access by a processor must be coordinated with the CU and wait for all other processors to come to a synchronization point. These wait and synchronization events can be a degradation in performance to a computation. The TN application is for multidimensional data manipulation, matrix processing, and data sorting, and can also perform a perfect shuffle. Unlike other more complicated and powerful permutation networks, the TN cannot, if possible at all, unscramble non-p-ordered vectors in one cycle.

I. INTRODUCTION

In the preliminary studies (refs. 1, 2), the Burroughs Corporation proposed a baseline computer system for the flow model processor (FMP) of the NASF. This proposed computer system is similar to the ILLIAC IV (ref. 3) parallel computer except that a few innovative ideas were incorporated. One of these innovations, the TN, is a bidirectional programmable combinational logic network that can be used to perform conflict-free access to various slices of multidimensional data (such as rows, columns, diagonals, and files) and subsequent transposition of these data for processing. The end result is that the TN provides a very simple and efficient method for the FMP to store its data in memory and allows parallel algorithms to be executed at a rate matched to the array processor rate.

The TN is one method of solving the traditional memory-processor connection problem in parallel array processors, but there are others (refs. 4 through 7). The tradeoffs in the design of memory-processor connection networks are basically complexity versus flexibility. A full crossbar switch, which is very complex and costly, can unscramble any data permutation and thus allow the system to store data without regard to subsequent unscrambling requirements. The TN, which is a low-cost and simple network, cannot unscramble every data permutation in one cycle. Therefore, because several cycles are needed to unscramble the data, certain data storage allocations can result in processing delay.

In the FMP, the TN connects an array of $N = 521$ memory modules, called the extended memory (EM), to an array of $R = 512$ processors, where N is selected as the smallest prime number greater than R. This memory-processor connection is shown in figure 1, which is a block diagram of the FMP. For certain types of data storage allocations in memory, such as the IBM FORTRAN IV method that requires that multidimensional data be stored in column major order, p-ordered vectors arise naturally from these storage allocations. A p-ordered vector, as defined by Swanson (ref. 8), is an n-element vector in which the $(i + 1)$th element is spaced p positions to the right of the ith element modulo N. The primary purpose of the TN is to transpose (or unscramble) p-ordered vectors to 1-ordered vectors in one network cycle. For unscrambling quasi-p-ordered vectors (called pq-ordered vectors), the TN speed is degenerated to several network cycles.

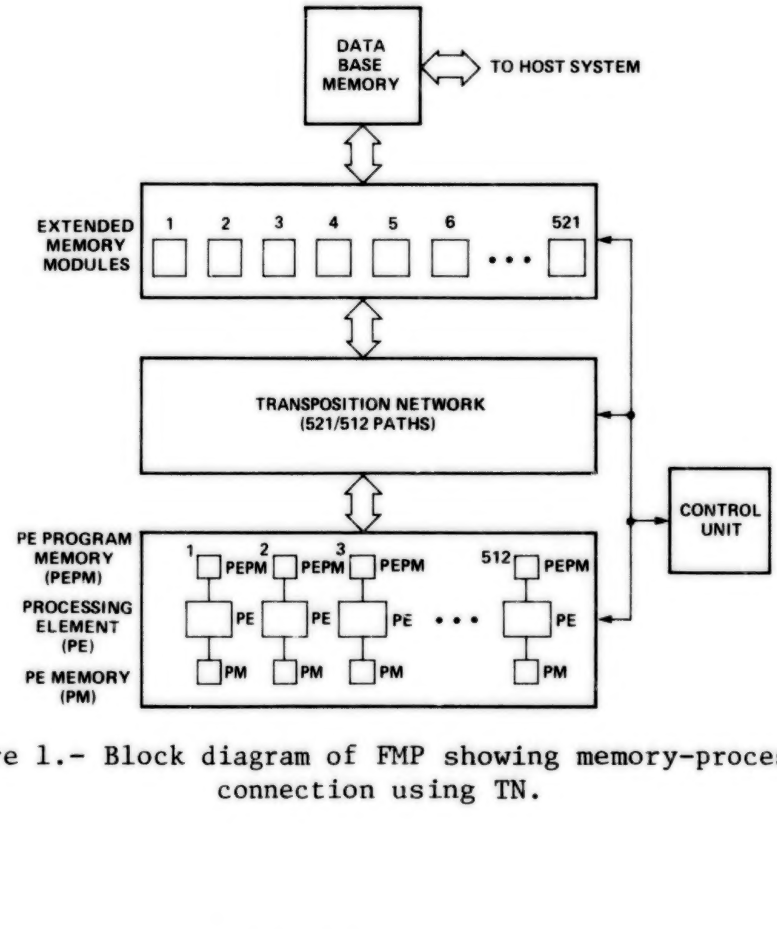


Figure 1.- Block diagram of FMP showing memory-processor connection using TN.

The design concept of TN is based upon Swanson's paper (ref. 8) concerning the use of k-apart interconnection networks to unscramble p-ordered vectors. In his paper, Swanson shows that a p-ordered vector can be unscrambled to a 1-ordered vector by using a k-apart network requiring $m \leq N - 2$ routings if k is a primitive root of the prime N, m is the index of p relative to k, and p is an element of the cyclic group M_N generated by k. G. Barnes of the Burroughs Corporation (ref. 1) observed that these $N - 2$ routings can be reduced to $L = \text{Log}_2 N$ routings by using L levels of k^v-apart networks where $v = 2^i$ and $i = 0,1,2, . . ., L - 1$. Furthermore, he observed that these L networks can be implemented by one level of barrel switch plus a fixed wiring pattern and its inverse. The connection of this fixed wiring pattern is from p to m according to $k^m \equiv p \pmod{N}$. The result is a very high-speed and low-cost TN, but with limited performance when compared to any other permutation networks presently known. The programming of the TN is simple, requiring only 20 bits: 10 bits for offset control and 10 bits for barrel switch shift control.

In this paper, a tutorial description of the TN is presented from the viewpoints of design, programming, and application in seven sections. Section II gives a brief description of the TN in the FMP and a description of p-ordered and pq-ordered vectors. Section III reviews cyclic groups from abstract algebra and primitive roots and indices from number theory. Section IV gives a summary of Swanson's paper. Section V describes the TN design using $\text{Log}_2 N$ k-apart interconnection networks. Section VI describes the TN design using one level of barrel switch. Finally, section VII describes TN programming and application.

II. p AND pq-ORDERED VECTORS

This section begins with a brief description of the TN and the FMP, and then shows how two- and three-dimensional data might be stored in such an architecture. There are many methods for storing data in multimodule memory systems (refs. 9 through 11). In this paper, only the FORTRAN column major order method is of interest because the FMP will use an extended FORTRAN as its high-level programming language. For the column major order method, it will be seen that p-ordered and pq-ordered vectors arise naturally from the FMP storage structure. The definition of p-ordered and pq-ordered vectors depends on the number of memory modules, and it is found that there are some advantages in storing data and unscrambling fetched vectors when the number of memory modules is restricted to be a prime number.

The FMP block diagram is shown in figure 1. This FMP architecture is similar to the ILLIAC IV except that a TN is used to connect 521 memory modules to 512 processors. The exact interconnection pattern is programmable and is controlled by the CU. There are no connections among the 512 processors, and any data exchanges between processors must be done by way of the EM using the TN. Once the CU sets up a particular interconnection pattern in the TN, a single read (or write) instruction by each processor will fetch (or store) a word from (to) its connected EM module. The net result is that a vector of data can be fetched (or stored) in one memory access. The fetched vector from EM, prior to the TN unscramble, can be a p-ordered, a pq-ordered, or any permuted vector. In the following, the concept and the definition of p-ordered and pq-ordered vectors are described. Instead of using $R = 512$ and $N = 521$ for the description, a simpler model consists of $R = 8$ and $N = 11$ will be used. Note that 11 is the smallest prime number greater than 8.

Let $X = (X_0\ X_1\ X_2\ \ldots\ X_{n-2}\ X_{n-1})$ be a one-dimensional vector, or simply just a vector, with n elements. Let X_i, $i = 0,1,2, \ldots, n - 1$ denote the ith element of X. In a computer, X is normally stored in N registers. In this case, one can say that associated with X_i is a position number j, $j = 0,1,2, \ldots, N - 1$, where $N \geqslant n$. For $N = n$, X_i is assigned as follows:

$$
\begin{array}{lccccccc}
j: & 0 & 1 & 2 & 3 & 4 \ \ldots & N-2 & N-1 \\
X_i: & X_0 & X_1 & X_2 & X_3 & X_4 \ \ldots & X_{n-2} & X_{n-1}
\end{array}
$$

Swanson defines a p-ordered vector X as an n-element vector, where X_{i+1} is spaced (or skipped) p positions to the right of X_i modulo N. In this paper, p is called the skip distance of X. In view of this definition, the above assignment of X_i to j is a 1-ordered vector because X_{i+1} is spaced one position to the right of X_i modulo N. Other examples of p-ordered vectors are illustrated below:

Example 2-1. $N = 11$, $n = 11$, $p = 3$, offset = 0

$$
\begin{array}{lccccccccccc}
j: & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
X_i: & X_0 & X_4 & X_8 & X_1 & X_5 & X_9 & X_2 & X_6 & X_{10} & X_3 & X_7
\end{array}
$$

Example 2-2. $N = 11$, $n = 8$, $p = 5$, offset = 2, * = don't care

$$
\begin{array}{lccccccccccc}
j: & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
X_i: & X_4 & X_2 & X_0 & * & X_7 & X_5 & X_3 & X_1 & * & * & X_6
\end{array}
$$

In the two examples above, the offset is defined as the location of the first element (X_0) of the n-element vector with respect to the first position of the register (or memory module) array. With these examples, Swanson's definition (ref. 8, p. 1107) for a p-ordered vector can be stated.

Definition: Let $j = 0,1,2, \ldots, N - 1$, $p = 1,2,3, \ldots, N - 1$, $i = 0,1,2, \ldots, n - 1$, $N \geqslant n$, and offset = 0. An n-element vector X is p-ordered if the positions j of its elements X_i are described by

$$pi \pmod{N} = j \tag{2-1}$$

From linear congruence in number theory, equation (2-1) can be written as

$$pi \equiv j \pmod{N} \qquad (2\text{-}2)$$

and there exists a unique solution for the unknown i for all values of j if and only if p is relatively prime to N (ref. 12, p. 84). Note that if N is prime, then all p-ordered vectors can be defined for $p = 1,2, \ldots, N - 1$. In the FMP, there are 512 processors. Therefore, the number of EM modules N is selected to be 521, which is the smallest prime number greater than 512. As a result, there is more flexibility in the manner in which the data can be stored, since all p-ordered vectors can be unscrambled.

Now that a p-ordered vector is defined, let's examine how p-ordered vectors arise naturally from data stored in memory modules. Consider the storage of the 4 × 4 × 4 three-dimensional data set of figure 2 in $N = 11$ memory modules, as shown in figure 3. In figure 2, an element in a plane, say a_{231}, is simply written as 231. In figure 3, M_j is used to denote the memory modules for $j = 0,1,2, \ldots, 10$, and i is used to denote the memory address within a module. Also, assume that the computer system has eight processors and all are assigned to compute on this data set. The memory accessing requirement by each processor will be described later in section VII, whereas in this section, only the illustration of p-ordered and pq-ordered vectors (as defined later) is of interest.

The natural formation of p-ordered vectors can be seen when the processors are computing in the k direction. In this direction, data from the IJ planes, one plane at a time, are required. As can be seen from figures 2 and 3, the fetching of any column or any two consecutive columns i and $i + 1$ are 1-ordered vectors with different offsets with respect to M_0, the first memory module. This is the case because the data are stored in column major order. As an example, the fetched vector consisting of columns 3 and 4 from

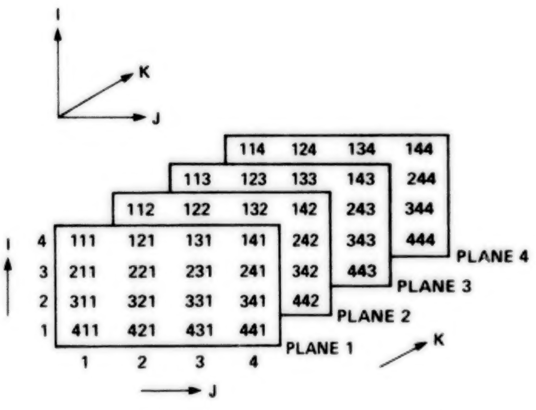


Figure 2.- A 4 × 4 × 4 three-dimensional data set.

i \ M_j	0	1	2	3	4	5	6	7	8	9	10
0	111	211	311	411	121	221	321	421	131	231	331
1	431	141	241	341	441	112	212	312	412	122	222
2	322	422	132	232	332	432	142	242	342	442	113
3	213	313	413	123	223	323	423	133	233	333	433
4	143	243	343	443	114	214	314	414	124	224	324
5	424	134	234	334	434	144	244	344	444		

Figure 3.- Column major order storage of a 4 × 4 × 4 data set in 11 memory modules, M_j is memory module number and i is a location within a memory module.

plane 1 is a 1-ordered vector with offset = 8 (the leading element is 131) and is shown below:

M_j:	0	1	2	3	4	5	6	7	8	9	10
X_i:	431	141	241	341	441	*	*	*	131	231	331

The fetched vector of any row in any plane is a 4-ordered vector because the number of elements in a column is four, which is the separation distance of the row elements in a column major order storage. As an example, the fetched vector of row 2 in plane 1 is a 4-ordered vector with offset = 1 (the leading element is 211) as shown below:

M_j:	0	1	2	3	4	5	6	7	8	9	10	
X_i:	*	211	241	*	*	221	*	*	*	231	*	row 2
X_i:	111	141	*	*	121	*	*	*	131	*	*	row 1

However, if an attempt is made to fetch both rows 1 and 2 from plane 1 simultaneously, there is a memory access conflict in memory module M_1 because elements 211 and 141 are both stored in M_1. In this case, the row vector consisting of rows 1 and 2 must be formed by two memory fetches through the TN. In general, memory access conflict exists if the memory module address of two elements is equal. Section VII gives equations for calculating element addresses in a three-dimensional data set and also describes how the skip distance p can be calculated.

The concept of a pq-ordered vector is derived from the concept of a p-ordered vector. The natural formation of pq-ordered vectors can be seen when the processors are computing in the J direction. In this direction, data from the IK planes, one plane at a time, are required. Referring to figures 2 and 3, the fetched vector consists of column 3 = $(142\ 242\ 342\ 442)^T$ and column 4 = $(141\ 241\ 341\ 441)^T$ in plane 4 is a pq-ordered vector with p = 1, q = 2, and offset = 6, as shown below. Also shown below is a pq-ordered vector with p = 7 and q = 5.

M_j:	0	1	2	3	4	5	6	7	8	9	10	p	q
X_i:	*	$\underline{141}$	$\underline{241}$	$\underline{341}$	$\underline{441}$	*	$\underline{142}$	$\underline{242}$	$\underline{342}$	$\underline{442}$	*	1	2
X_i:	$\underline{431}$	422	*	$\underline{411}$	$\underline{441}$	432	*	$\underline{421}$	412	442	*	7	5

This vector is a result of fetching column 1 = $(441\ 431\ 421\ 411)^T$ and column 2 = $(442\ 432\ 422\ 412)^T$ from plane 1 when computing in the I direction. For unscrambling pq-ordered vectors, the TN cannot perform the task in one cycle. In the above examples, two memory fetches and unscramblings are required. In general, the total number of memory fetches and unscramblings is equal to the number of groups within the vector X. Each group within X is a p-ordered vector, and the separation distance between groups is q. For this reason, pq-ordered vector fetching sometimes is called periodic fetching by groups.

The concept and natural formation of p-ordered and pq-ordered vectors have been presented in this section. The 4 × 4 × 4 three-dimensional data set in figure 2 and its memory module storage allocation scheme shown in figure 3 are only for illustrative purposes. In actual practice, the data set and the storage allocation scheme may be different. The purpose here is to conjecture that the TN, as described later in the design sections, can be used to unscramble a p-ordered vector to a 1-ordered vector in one cycle and to unscramble a pq-ordered vector to a 1-ordered vector in several cycles. The exact number of cycles is dependent on such factors as data storage allocation schemes, memory access conflicts, and the number of periodic groups within a vector.

III. CYCLIC GROUPS, PRIMITIVE ROOTS, AND INDICES

In the design of the TN, the concept of cyclic groups, primitive roots, and indices are used. For this reason, a brief review of these concepts necessary to understand the TN design is presented. In summary, a primitive root in number theory is a special case of a generator of a cyclic group in abstract algebra. The theory of primitive roots and indices can be used to solve certain types of congruence equations in number theory. The reader, if so desired, can skip this section and proceed to the next section where the description of the Swanson network is presented.

Cyclic Groups

From abstract algebra, a set is a collection of elements with some common properties that can be used to determine whether an element does or does not belong to the set. A group G is a nonempty set of elements and a binary operation on the set (i.e., the set is closed under the group operation) such that the operation is associative, there is a unique identity element for the operation, and each element has an inverse element for the operation. If the operation is commutative, then G is called a commutative (or abelian) group. If the number of elements in G is finite, then G is called a finite group. Otherwise, G is called an infinite group. The number of elements contained in G is called the order of G. The binary operation of G is often written as multiplication. If the operation is written as multiplication, then G is called a multiplicative group. In the design of the TN, only finite multiplicative groups will be used.

Let g be an element of G and e be the identity element of G. The order of g is the smallest positive integer t such that $g^t = e$. Note that if G is finite, every element g has an order and the order of g divides the order of G.

A group G is called cyclic if it contains an element g such that every element h of G can be expressed as

$$h = g^{\ell} \qquad (3\text{-}1)$$

for some integral exponent ℓ, positive, negative, or zero. Such an element g of a cyclic group G is called a generator of G. Note that g is a generator of G (and G is cyclic) if and only if the order of g is equal to the order of G. If G has N - 1 elements generated by g modulor N, then the generator g is called a primitive element of G, or a primitive generator. The following examples will illustrate the concept.

Example 3-1. The number 3 is a generator of the cyclic group G = (1,2,3,4,5,6) under multiplication modulo 7, since $3^1 = 3$, $3^2 = 2$, $3^3 = 6$, $3^4 = 4$, $3^5 = 5$, $3^6 = 1$

Example 3-2. The number 2 is a generator of the cyclic group G = (1,2,3,4,5,6,7,8,9,10) under multiplication modulo 11, since $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 5$, $2^5 = 10$, $2^6 = 9$, $2^7 = 7$, $2^8 = 3$, $2^9 = 6$, $2^{10} = 1$

Example 3-3. The number 4 is a generator of the cyclic group G = (1,2,4) under multiplication modulo 7, since $4^1 = 4$, $4^2 = 2$, $4_3 = 1$

Let N be a positive integer and let M_N consist of all the positive integers that are less than N and relatively prime to N. Then M_N forms a group under the binary operation multiplication modulo N. If N is prime, then M_N contains all the integers from 1 to N - 1 and, therefore, has order N - 1. Furthermore, if N is prime, M_N is cyclic and thus has at least one generator. In the above two examples, M_7 and M_{11} are two such cyclic groups generated by 3 and 2, respectively. In the TN design, N is chosen to be 521 and 3 (the smallest positive integer generating M_{521} and also a primitive root of 521 as defined later) is chosen as the primitive generator to generate all the integers from 1 to 520 modulo 521. This is to ensure that all p-ordered vectors, for p = 1,2, . . ., 520, can be unscrambled in one cycle. The design of the Swanson network is based upon the concept of cyclic groups and linear congruences as described later.

Primitive Roots

The concept of primitive roots is a special case of the concept of cyclic group generators. Let a group G have N - 1 elements. If g of G is a primitive generator, then g is a primitive root of N. As previously described, the discussion of cyclic groups is a topic in group theory in abstract algebra (refs. 13 and 14), whereas the discussion of primitive roots is a topic in number theory (refs. 12, 15, and 16). A generator in G is one that generates a cyclic group (or subgroup) of G. The generator that generates a cyclic group of order N - 1 modulo N is called a primitive element (or primitive root) of G. Thus, primitive generators and primitive roots are the same. In this section, the concept of primitive roots from number theory is briefly reviewed, starting with Euler's theorem.

In number theory (refs. 12, 15, and 16), Euler's phi-function $\phi(N)$, for $N \geq 1$, denotes the number of positive integers not exceeding N that are relatively prime to N. If N is a prime number, then every positive integer less than N is relatively prime to it; whence $\phi(N) = N - 1$. On the other hand, if $N > 1$ is composite, then N has a divisor d such that $1 < d < N$. It follows that there are at least two integers among $1, 2, 3, \ldots, N$ that are not relatively prime to N, namely, d and N itself. As a result, $\phi(N) \leqslant N - 2$. This proves: for $N > 1$,

$$\phi(N) = N - 1 \text{ if and only if } N \text{ is prime} \tag{3-2}$$

If N is not prime, the value of $\phi(N)$ can be obtained either analytically or by table lookup (ref. 17). In reference 17 (pp. 840 to 843), $\phi(N)$ is listed for N ranging from 1 to 1000. One useful property of $\phi(N)$ is given below without proof.

<u>Euler's theorem</u>: If a and N are positive integers and $\gcd(a, N) = 1$, then $a^{\phi(N)} \equiv 1$ modulo N.

In view of Euler's theorem, it is known that $a^{\phi(N)} \equiv 1$ modulo N whenever $\gcd(a, N) = 1$. However, there are often powers of a smaller than $a^{\phi(N)}$ that are congruent to 1 modulo N. This leads to the definition of the order of a modulo N (in older terminology: the exponent to which a belongs modulo N):

<u>Order definition</u>: Let $N > 1$ and $\gcd(a, N) = 1$. The order of a modulo N is the smallest positive integer k such that $a^k \equiv 1$ modulo N.

As an example, consider the successive powers of 2 modulo 7 as follows:

$$2^1 \equiv 2, \quad 2^2 \equiv 4, \quad 2^3 \equiv 1, \quad 2^4 \equiv 2, \quad 2^5 \equiv 4, \quad 2^6 \equiv 1, \ldots$$

from which it follows that the integer 2 has order 3 modulo 7. This example also shows that $2^k \equiv 1$ modulo 7 whenever k is a multiple of 3. This leads to a theorem for finding the order of an integer a.

<u>Order theorem</u>: Let the positive integer a have order k modulo N. Let $h > k$, then $a^h \equiv 1$ modulo N if and only if k divides h; in particular, k divides $\phi(N)$.

This theorem expedites the computation of the order of an integer a modulo N. Instead of considering all powers of a, the exponents can be restricted to the divisors of $\phi(N)$.

The order theorem will lead directly into the definition of primitive roots. Note that the order k of an integer a modulo N is necessarily a divisor of $\phi(N)$. The largest of these divisors is $\phi(N)$ itself. Integers of order $\phi(N)$ modulo N are called primitive roots of N. Note that in general there may be no primitive roots of N. However, in the case that N is a prime, primitive roots of N always exist. The formal definition of primitive root is stated as follows:

<u>Primitive root definition</u>: If gcd $(a, N) = 1$ and a is of order $\phi(N)$ modulo N, then a is a primitive root of N.

In other words, N has a as a primitive root if $a^{\phi(N)} \equiv 1$ modulo N, but $a^k \not\equiv 1$ modulo N for all positive integers $k < \phi(N)$. Thus, a primitive root of the positive integer N is an integer that has the largest possible order. This means that the powers of a primitive root modulo N will generate all the integers not exceeding N. From this analysis, it can be seen that primitive roots are a special case of generators (primitive) of cyclic groups. As indicated earlier in example 3-2 (cyclic groups), 2 is a primitive generator for the cyclic group $G = (1,2,3,4,5,6,7,8,9,10)$ modulo 11. In number theory, 2 is a primitive root of 11 because $\phi(11) = 10$, $2^{10} \equiv 1$ modulo 11, $2^k \not\equiv 1$ modulo 11 for $k < 10$, and thus the powers of 2 generate all the elements of G.

In the study of number theory today, it is not known whether or not there exist primitive roots for all integers. However, it is known that a positive integer N has primitive roots if and only if N is 2,4, a power of an odd prime, or twice a power of an odd prime. Also, there is no known simple and efficient algorithm to find the primitive roots other than by the use of the definition. In general, if a is a primitive root of N, then any other primitive root of N is found among the member of the set $(a, a^2, \ldots a^{\phi(N)})$. It is known, however, that primitive roots do exist, the exact number of primitive roots are given by the following theorems, starting with the congruence theorem first.

<u>Congruence theorem</u>: Let gcd $(a, N) = 1$ and let $a_1, a_2, \ldots a_{\phi(N)}$ be the positive integers less than N and relatively prime to N. If a is a primitive root of N, then $a, a^2, \ldots, a^{\phi(N)}$ are congruent modulo N to $a_1, a_2, \ldots, a_{\phi(N)}$ in some order.

The congruence theorem leads immediately to the next two theorems.

<u>Primitive root theorem</u>: If N has a primitive root, then it has exactly $\phi[\phi(N)]$ of them.

<u>Prime primitive root theorem</u>: If N is a prime, then there are exactly $\phi(N - 1)$ incongruent primitive roots of N.

The above theorems will be illustrated by two examples.

Example 3-4. Let $N = 11$. There are exactly $\phi(10) = 4$ primitive roots of 11. The smallest primitive root of 11 is 2 (by table lookup). The other three primitive roots of 11 must be among the member of the set 2^i, $i = 1,2, \ldots, 10$ modulo 11 as shown below:

$$2^1 \equiv 2, \quad 2^2 \equiv 4, \quad 2^3 \equiv 8, \quad 2^4 \equiv 5, \quad 2^5 \equiv 10$$

$$2^6 \equiv 9, \quad 2^7 \equiv 7, \quad 2^8 \equiv 3, \quad 2^9 \equiv 6, \quad 2^{10} \equiv 1$$

By tedious calculation, these three primitive roots of 11 are 6, 7, and 8. The 10 powers of the primitive root 2 presented above formed the cyclic group M_{11} and therefore are congruent in some order to the $\phi(11)$ numbers in the set $(1,2,3, \ldots, 10)$.

Example 3-5. Let $N = 9$, which is not a prime number. There are exactly $\phi(\phi(9)) = \phi(6) = 2$ primitive roots of 9. The smallest primitive root of 9 is 2 (by table lookup). The other primitive root can be found among the six powers of 2 modulo 9 as shown below:

$$2^1 \equiv 2, \quad 2^2 \equiv 4, \quad 2^3 \equiv 8, \quad 2^4 \equiv 7, \quad 2^5 \equiv 5, \quad 2^6 \equiv 1$$

By tedious calculation, this primitive root of 9 is 5. Now the integers less than and relatively prime to 9 are 1, 2, 4, 5, 7, and 8, and these numbers are congruent in some order to the six powers of 2 presented above.

In the TN design, a prime number $N = 521$ is selected. There are $\phi(520) = 192$ primitive roots to choose from as a primitive generator to generate the cyclic group $M_{521} = (1,2, \ldots, 520)$. In practice, the smallest primitive root is selected as the generator. For $N = 521$, 3 is the smallest primitive root. Tables of the smallest primitive root r of the prime number p can be found in references 12 and 17. In reference 12 (p. 327), r is given for $2 \leqslant p < 1000$. In reference 17 (pp. 864 to 869), r is given for $3 \leqslant p \leqslant 9973$. Other than these tables, there are no known algorithms today, for example, to find the 192 primitive roots of $N = 521$ other than essentially using the stated definition. It is to be noted, however, that 2 is not a primitive root of $N = 521$ because $2^{260} \equiv 1$ modulo 521, which violates the primitive root definition, since $260 < \phi(521) = 520$. A method of calculating 2^{260}, an old and undocumented method in number theory, is shown in appendix A. As will be described later, this method also will be used as the first high-speed implementation method of the Swanson network.

Theory of Indices

The theory of indices is an old and neglected topic in number theory. It is analogous to that of logarithms, where the primitive root plays the part similar to that of a base of a logarithm. The major difference is that indices are computed using the theory of congruences. In the TN design, the barrel switch implementation of the Swanson network can be described most simply by using the theory of indices.

Let N be a positive integer. Two integers a and b are said to be congruence modulo N, symbolized by

$$a \equiv b \pmod{N} \tag{3-3}$$

if N divides the difference $a - b$. Let r be the primitive root of N. From the congruence theorem, the first $\phi(N)$ powers of r,

$$r, r^2, \ldots, r^{\phi(N)}$$

are congruent modulo N, in some order, to those integers less than N and relatively prime to it. Hence, if a is an arbitrary integer relatively prime to N, then a can be expressed in the form

$$a \equiv r^k \pmod{N} \tag{3-4}$$

for a suitable choice of k, where $1 \leqslant k \leqslant \phi(N)$. The exponent k is called the index of a relative to r, and this leads to the formal definition on the concept of index.

Index definition: Let r be a primitive root of N. If $\gcd(a,N) = 1$, then the smallest positive integer k such that $a \equiv r^k$ modulo N is called the index of a relative to r.

The standard notation for the index of a relative to r is $\text{ind}_r a$ or, if no confusion is likely to occur, ind a is used. Clearly, $1 \leqslant \text{ind}_r a \leqslant \phi(N)$ and

$$r^{\text{ind}_r a} \equiv a \pmod{N} \tag{3-5}$$

Note that the definition of index is meaningless unless $\gcd(a,N) = 1$. The following example will illustrate the concept of index.

Example 3-6. The integer 2 is a primitive root of 11 and the 10 powers of 2 are listed in example 3-4. From this list a table of indices can be prepared as follows:

a	1	2	3	4	5	6	7	8	9	10
$\text{ind}_2 a$	10	1	8	2	4	9	7	3	6	5

It follows that

$\text{ind}_2 1 = 10,\ \text{ind}_2 2 = 1,\ \ldots,\ \text{ind}_2 10 = 5$

With indices, as with logarithms, multiplication, division, evolution, and involution are replaced by addition, subtraction, multiplication, and division, respectively. These concepts are summarized in the following theorem without proof.

Index theorem 1: If N has a primitive root r and ind a denotes the index of a relative to r, then

(1) $\text{ind } (ab) \equiv \text{ind } a + \text{ind } b$ modulo $\phi(N)$

(2) $\text{ind } a^k \equiv k \text{ ind } a$ modulo $\phi(N)$ for $k > 0$

(3) $\text{ind } 1 \equiv 0$ modulo $\phi(N)$, $\text{ind } r \equiv 1$ modulo $\phi(N)$

The theory of indices can be used to solve certain types of congruences. For example, consider the binomial congruence

$$X^k \equiv a \text{ modulo } N \, , \qquad k \geqslant 2 \tag{3-6}$$

where X is the unknown, a is an integer with $\gcd(a,N) = 1$, and assume N has r as a primitive root. By property (2) of index theorem 1, this congruence is equivalent to the linear congruence

$$k \text{ ind } X \equiv \text{ind } a \text{ modulo } \phi(N)$$

If $d = \gcd(k,\phi(N))$ and $d \times \text{ind } a$, there is no solution. But if $d \mid \text{ind } a$, then there are exactly d incongruent solutions. This leads to the following two theorems.

Index theorem 2: Let N be an integer with primitive root r and $\gcd(a,N) = 1$. Then the congruence $X^k \equiv a$ modulo N has a solution if and only if

$$a^{\phi(N)/d} \equiv 1 \text{ modulo } N$$

where $d = \gcd(k,\phi(N))$; if it has a solution, there are exactly d solutions modulo N.

Index theorem 3: Let N be a prime and $\gcd(a,N) = 1$. Then the congruence $X^k \equiv a$ modulo N has a solution if and only if

$$a^{(N-1)/d} \equiv 1 \text{ modulo } N$$

where $d = \gcd(k,N - 1)$.

To conclude this section, two examples are given below to illustrate these concepts.

Example 3-7. Solve the binomial congruence

$$7X^3 \equiv 3 \pmod{11}$$

A table of indices using the primitive root $r = 2$ is presented in example 3-6. The gcd $(k, \phi(N))$ = gcd $(3,10) = 1$, and 1 divides $\text{ind}_2 a = \text{ind}_2 3 = 8$. So there is exactly one solution. This solution is found as follows:

$$\text{ind}_2 7 + 3\ \text{ind}_2 X \equiv \text{ind}_2 3 \pmod{10}$$

$$7 + 3\ \text{ind}_2 X \equiv 8 \pmod{10}$$

$$3\ \text{ind}_2 X \equiv 1 \pmod{10}$$

At this point X must be found by direct computation. By direct substitution of $\text{ind}_2 X$, for $X = 1,2,3, \ldots, 10$ into the last congruence, only $X = 7$ is satisfied. Thus $X = 7$ is the solution.

Example 3-8. Solve the binomial congruence

$$3X^4 \equiv 9 \pmod{11}$$

The gcd = $d = (4,10) = 2$ and $d \mid \text{ind}_2 9 = 6$, so there are two solutions. Now,

$$\text{ind}_2 3 + 4\ \text{ind}_2 X \equiv \text{ind}_2 9 \pmod{10}$$

$$8 + 4\ \text{ind}_2 X \equiv 6 \pmod{10}$$

$$4\ \text{ind}_2 X \equiv -2 \pmod{10}$$

$$\equiv 8 \pmod{10}$$

Since gcd $(4,8,10) = 2$, then

$$2\ \text{ind}_2 X \equiv 4 \pmod{5}$$

Since gcd $(2,5) = 1$, then

$$\text{ind}_2 X \equiv 2 \pmod{5}$$

or

$$\text{ind}_2 X \equiv 2,7 \pmod{5}$$

The solutions are $X = 4$ and 7.

IV. SWANSON NETWORK

The Swanson network (ref. 8) is a simple interconnection network that can be used to unscramble p-ordered vectors. Its design is based on a so-called k-apart interconnection network where k is a primitive root of N, and N is the number of memory modules or the number of registers in a register array, depending on the context of the description. Because of its simplicity, the Swanson network requires m iterative routings to unscramble a p-ordered vector. The value m is related to k and p by $k^m \equiv p \pmod N$. If N is prime, then at most $m = N - 2$ routings are required.

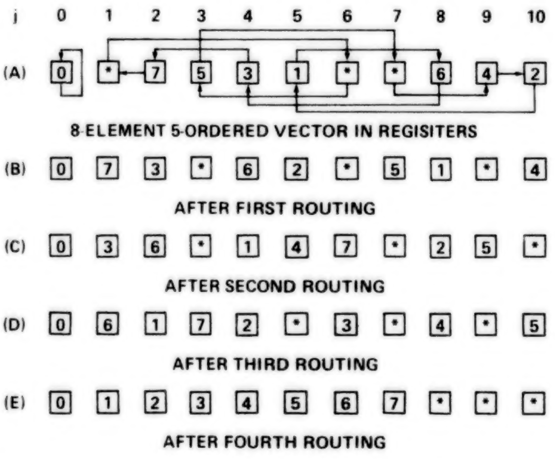
Let the n-element vector X, after it is fetched from a memory system consisting of N modules, be stored in N registers, where obviously $n \leqslant N$. Let the elements of X be X_i, $i = 0,1,2, \ldots, n - 1$. Let j be the positional index of N, and $j = 0,1,2, \ldots, N - 1$. The vector X is p-ordered if the positions of its elements X_i are described by

$$p\ i \equiv j \pmod N \qquad (4\text{-}1)$$

Equation (4-1) is the same as equation (2-2) as described previously in section II. A 5-ordered vector for $n = 8$ and $N = 11$ is shown in figure 4(a). From this definition of p-ordered vector, it is known from number theory that the linear congruence in equation (4-1) has a unique solution for i if and only if $\gcd(p,N) = 1$, and the equation can be solved by the theory of primitive roots and indices. For this reason, Swanson proposed an interconnection called a k-apart interconnection network and is defined as follows:

<u>Definition</u>. N registers are interconnected with a k-apart interconnection, $k = 1,2, \ldots, N - 1$, if the content of register $kj \pmod N$ can be transferred directly to register j. The notation for such an interconnection is

$$\text{Reg } (kj \bmod N) \rightarrow \text{Reg } (j) \qquad (4\text{-}2)$$



j 0 1 2 3 4 5 6 7 8 9 10
(A) 0 • 7 5 3 1 • • 6 4 2
8-ELEMENT 5-ORDERED VECTOR IN REGISITERS
(B) 0 7 3 • 6 2 • 5 1 • 4
AFTER FIRST ROUTING
(C) 0 3 6 • 1 4 7 • 2 5 •
AFTER SECOND ROUTING
(D) 0 6 1 7 2 • 3 • 4 • 5
AFTER THIRD ROUTING
(E) 0 1 2 3 4 5 6 7 • • •
AFTER FOURTH ROUTING

Figure 4.- Unscrambling a 5-ordered vector with a 2-apart interconnection of 11 registers.

The problem confronted now is to determine the value of k. As with the definition of p-ordered vectors, k must be restricted to be relatively prime to N so that all p-ordered vectors can be unscrambled in, say, m routings. When a vector contained in the registers is to be routed along the interconnection paths, all registers transfer their content simultaneously. If the registers contain a p-ordered vector with $p = k$, then one transfer is sufficient to unscramble the vector to a 1-ordered vector. In order to meet these requirements, k is necessarily a primitive root of N as will be described later. With this information, an interconnection network for $N = 11$ can be constructed using $k = 2$ since 2 is a primitive root of 11. From equation (4-2), the calculation for the connections is shown in table 1, and the physical connections are shown in figure 4(a). The unscrambling of a 5-ordered vector requires $m = 4$ routings, and this is calculated from

TABLE 1.- INTERCONNECTION FOR 2-APART NETWORK (MOD 11)

j	2j (mod 11)	Connection
0	0	$0 \rightarrow 0$
1	2	$2 \rightarrow 1$
2	4	$4 \rightarrow 2$
3	6	$6 \rightarrow 3$
4	8	$8 \rightarrow 4$
5	10	$10 \rightarrow 5$
6	1	$1 \rightarrow 6$
7	3	$3 \rightarrow 7$
8	5	$5 \rightarrow 8$
9	7	$7 \rightarrow 9$
10	9	$9 \rightarrow 10$

$$2^m \equiv 5 \pmod{11} , \quad m = 4$$

The step-by-step process of this unscrambling is shown in figure 4. It is interesting to note how p changes after each routing. Since it requires a total of four routings, the progression of p with respect to m is

$$\begin{array}{llllll} m = & 0 & 1 & 2 & 3 & 4 \\ p = & 5 & 8 & 4 & 2 & 1 \end{array} \qquad (4\text{-}3)$$

which is a progression of the powers of $k = 2$. After the $(m - 1)$th routing, the vector always reduces to k-ordered; thus, a p-ordered vector, with $p = k$, can be unscrambled in one more routing.

From the discussions of p-ordered vectors in section II and cyclic groups in section III, N is necessarily a prime so that all p-ordered vectors, for $p = 1,2,3, \ldots, N - 1$, can be unscrambled. Since k is a primitive root of N, then the interconnections generated by equation (4-2) are a cyclic group M_N of order $N - 1$. The resultant interconnection will unscramble any p-ordered vector in m routings according to

$$k^m \equiv p \pmod{N} \qquad (4\text{-}4)$$

With these discussions, a theorem can now be formulated.

Swanson theorem: If N registers are interconnected with a k-apart interconnection $K \in M_N$, then a p-ordered vector, $p \in M_N$, contained in these registers can be converted to a 1-ordered vector if and only if p is an element of the cyclic group M_N generated by k.

The proof of this theorem is simply the fact that after m routings through the network, the contents of the registers are given by

$$\text{Reg } [j] \equiv (p^{-1}k^m)\ j \pmod{N} \qquad (4\text{-}5)$$

for $j = 0,1,2, \ldots, N - 1$ and p^{-1} is the multiplicative inverse of p modulo N. In order for the vector to become 1-ordered after some number of routings, m from equation (4-5) must satisfy

$$\text{Reg } [j] \equiv j = (p^{-1}k^{m})\ j \pmod{N} \qquad (4\text{-}6)$$

A value for m satisfies this equation for all values of j if and only if m satisfies

$$p^{-1}k^{m} \equiv 1 \pmod{N} \qquad (4\text{-}7)$$

or

$$k^{m} \equiv p \pmod{N} \qquad (4\text{-}8)$$

From this, it is clear that the p-ordered vector can be unscrambled if and only if p is an element of the cyclic group M_N generated by k. Furthermore, k must necessarily be a primitive root of N so that all values of p, for $p = 1,2, \ldots, N - 1$, can be generated by equation (4-8). For an arbitrary value of p between 1 and $N - 1$, the maximum value of m in equation (4-8) is $N - 2$, which is the maximum number of routings required to unscramble a p-ordered vector. As an example (TN case), let $N = 521$ and $k = 3$, then $3^{519} \equiv 174 \pmod{521}$.

In summary, it can be seen from the theorem that if N is a prime and k is a primitive root of N, then M_N generated by k is a cyclic group. The order of M_N is $N - 1$ and the elements are $1,2, \ldots, N - 1$. All p-ordered vectors, for $p = 1,2, \ldots, N - 1$, can be unscrambled to a 1-ordered vector by a single k-apart interconnection network. This unscrambling process is effected by routing the elements of the vector through the network m times (see, e.g., fig. 4) according to

$$k^{m} \equiv p \pmod{N} \qquad (4\text{-}9)$$

The maximum number of routings is $m = N - 2$. After the $(m - 1)$th routing, the vector always reduces to k-ordered so that the vector can be reduced to a 1-ordered vector after the mth routing. Prior to the mth routing, the reduction of p after each routing is progressing in the powers of k (see, e.g., eq. (4-3)).

V. TRANSPOSITION NETWORK DESIGN: k-APART NETWORK IMPLEMENTATION

As described in section IV, the Swanson k-apart network can be used to unscramble an n-element p-ordered vector stored in N registers $n \leq N$, with a maximum of $N - 2$ routings. If N is large, like 521 in the FMP, then the unscrambling can be a long delay. To overcome this problem, several k-apart networks can be connected in series to perform the $N - 2$ routings in the progression of the powers of k as suggested in equation (4-3). In electrical engineering this technique is called logarithmic compression, which is similar to the technique of computing N^{k} in $2\log_2(k + 1)$ steps as shown in appendix A. In the sequel, it will be shown that the $N - 2$ routings can be reduced to $L = \log_2 N$ routings by using L k^{v}-apart networks, where $v = 2^{i}$ for $i = 0,1,2, \ldots, L - 1$.

Let the integer N be represented by an L-bit binary number. At most, when N is at its maximum value, N is

$$N = 2^{L-1} + . . . + 2^1 + 2^0 = 2^L - 1 \qquad (5\text{-}1)$$

or

$$N - 2 = 2^L - 3 \qquad (5\text{-}2)$$

This gives

$$L = \log_2 (N + 1)$$

$$\cong \lceil \log_2 N \rceil = \left\lceil \frac{\ln N}{\ln 2} \right\rceil \text{ for } N >> 1 \qquad (5\text{-}3)$$

for $N = 521$, $L = 10$, and for $N = 11$, $L = 4$. In equation (5-3), it is suggested that the $N - 2$ routings can be reduced to $L = \log_2 N$ routings by using L k-apart networks connected in series in L levels. At level i, the network will perform either a straight-through connection (no routing) or a routing by the amount equal to k^v (mod N), where $v = 2^i$ for $i = 0,1,2, . . ., L - 1$. The selection either "to route" or "straight through" is controlled by the binary bit 2^i of m, where m is calculated as

$$k^m \equiv p \pmod N \qquad (5\text{-}4)$$

Let U(x) denote the k-apart network at level i for $i = 0,1, . . ., L - 1$. Then the L levels of networks are as follows:

$U(x) = k^{2^0}$	(mod N)	1st level
$U(x) = k^{2^1}$	(mod N)	2nd level
.		
.		
.		
$U(x) = k^{2^{L-1}}$	(mod N)	Lth level

As an example, for $N = 11$, $k = 2$, the four levels of networks are:

U(2) = 2-apart	1st level
U(4) = 4-apart	2nd level
U(5) = 5-apart	3rd level
U(3) = 3-apart	4th level

Note that the combinations of the numbers 2, 3, 4, and 5 can be selected appropriately to form any number from 2 to 10; and therefore, it can be used to unscramble any p-ordered vector for $p = 1,2, . . ., 10$. For the trivial case when the vector to be unscrambled is a 1-ordered vector, then all four levels will be selected as "straight through," because m is zero in this case, as calculated by equation (5-4). This concept of using L levels of k-apart networks to implement the TN is shown in figure 5. The implementation of a TN for $N = 11$ and $k = 2$ is shown in figure 6 with m set to 9 for unscrambling a 6-ordered vector, since $2^9 \equiv 6 \pmod{11}$. For the case when $N = 521$ and $k = 3$, the 10 levels of networks are U(3), U(9), U(16), U(256), U(411), U(117), U(143), U(130), U(228), and U(405).

VI. TRANSPOSITION NETWORK DESIGN: BARREL SWITCH IMPLEMENTATION

The implementation of the Swanson Network using $L = \log_2 N$ k-apart networks is a significant step in increasing the unscrambling speed by reducing the number of routings m from $N - 2$ to $\log_2 N$. The cost for this gain in speed is the increase in hardware complexity, where the number of k-apart networks is increased from 1 to $\log_2 N$. There is, however, an alternative to the implementation. By a close inspection of the k-apart network, it can be observed that there is a uniform shift pattern in its input to output connections (see, e.g., fig. 6). For this reason, it is reasonable to expect that a uniform-shift network (ref. 18), commonly called a barrel switch, can be used for such an implementation. In the sequel, it will be shown that a single level of barrel switch, plus a fixed wiring pattern connected according to $k^m \equiv p \pmod N$ and its inverse, can be used to replace all $L = \log_2 N$

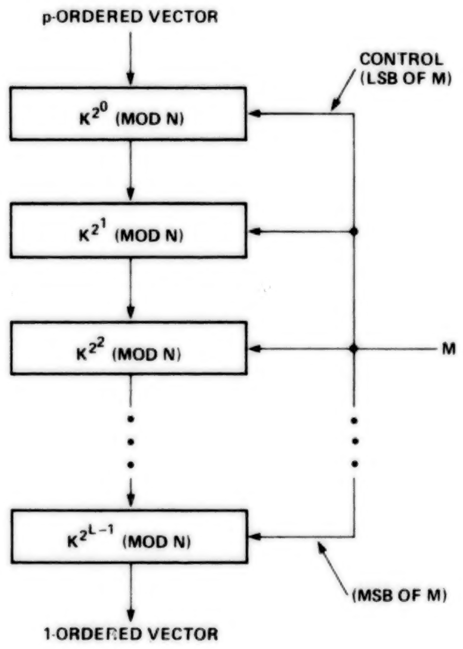


Figure 5.- Implementation of Swanson network using L levels of k-apart networks.

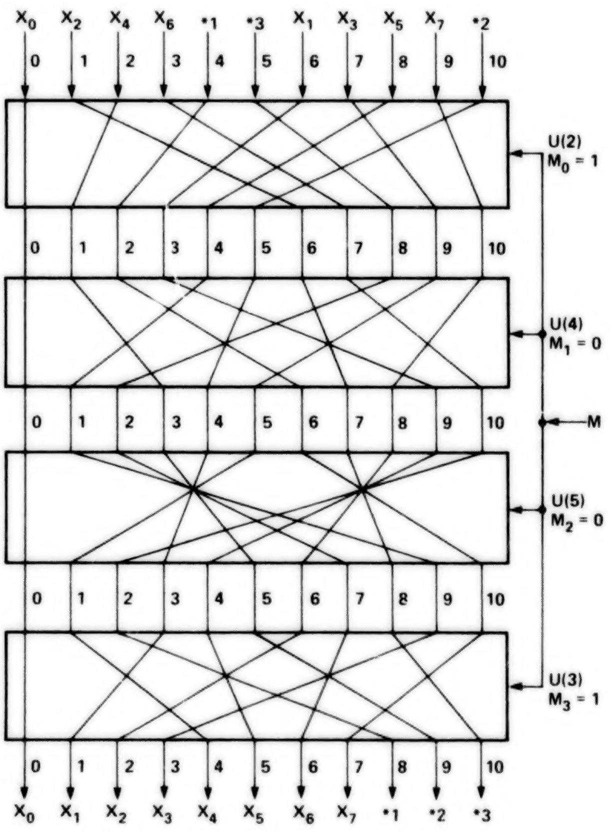

x_0 x_2 x_4 x_6 •1 •3 x_1 x_3 x_5 x_7 •2
0 1 2 3 4 5 6 7 8 9 10
U(2) $M_0 = 1$
U(4) $M_1 = 0$
M
U(5) $M_2 = 0$
U(3) $M_3 = 1$
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 •1 •2 •3

Figure 6.- TN for $N = 11$ and $k = 2$, showing the unscrambling of a 6-ordered vector for the routing amount $M = M_3M_2M_1M_0 = 1001$. A "0" means straight through and a "1" means routing.

levels of k-apart networks. The control for the barrel switch is simple where m simply controls the switch shift amount.

A barrel switch is a combinational logic network that can perform a uniform shift of an incoming vector by a fixed number of positions. The shift direction can be either left or right with options of either end-off or end-around. The design description of barrel switches is almost all contained in patent literature.

In "open" literature, design descriptions are few and far between. There are a few pages in the book by Burroughs Corporation in 1968 (ref. 19), the Signetic 8243 8-position scalar in 1970 (ref. 20), a few pages of description in 1970 by Davis (ref. 21), and a design note in 1972 by Lim (Ref. 22). In this section, barrel switch design is not of concern, and it will be treated simply as a uniform shift network, in particular, a left end-around uniform-shift network.

In the description of p-ordered vectors in section II, the vector X is p-ordered if the positions of its elements X_i are described by

$$X_i = j \equiv pi \pmod{N} \tag{5-1}$$

After X is unscrambled by the network, the result is a 1-ordered vector, or $p = 1$ in equation (5-1). That is,

$$X_i \equiv i \pmod{N} \tag{5-2}$$

or

$$X_i = i \tag{5-3}$$

for $i = 0,1,2, \ldots, n - 1$, and $n \leqslant N$. The objective of this section is to show that a Swanson network implemented by one row of barrel switch, plus a fixed unscrambling wiring pattern and its inverse, can be used to reduce equation (5-1) to equation (5-3).

From the Swanson k-apart network discussion in section IV,

$$k^m \equiv p \pmod{N} \tag{5-4}$$

where k is the primitive root of N and m is the number of routings required to unscramble p. Combining equations (5-1) and (5-4), the result is

$$i\, k^m \equiv X_i \pmod{N} \tag{5-5}$$

In the description of number theory in section III, equation (5-5) is a binomial congruence and it can be solved by the theory of indices. Taking indices on both sides of equation (5-5), the result is

$$\text{ind}_k i + m\ \text{ind}_k k \equiv \text{ind}_k X_i \pmod{\phi(N)} \tag{5-6}$$

or

$$\text{ind}_k i \equiv \text{ind}_k X_i - m\ \text{ind}_k k \pmod{\phi(N)} \tag{5-7}$$

since

$$k^m \equiv p \pmod{N}$$

then

$$m\ \text{ind}_k k = \text{ind}_k p \pmod{\phi(N)} \tag{5-8}$$

Combining equations (5-7) and (5-8), the result is

$$\text{ind}_k i \equiv \text{ind}_k X_i - \text{ind}_k p \pmod{\phi(N)} \tag{5-9}$$

or

$$\text{ind}_k i \equiv \text{ind}_k pi - \text{ind}_k p \pmod{\phi(N)} \tag{5-10}$$

Equation (5-10) is the key equation in demonstrating that the barrel switch indeed can implement the Swanson network. The implementation of this key equation is shown in figure 7. The explanation of figure 7 is as follows:

1. The input to the unscrambling wiring pattern W, point-A, is a p-ordered vector $X_i \equiv pi \pmod{N}$.

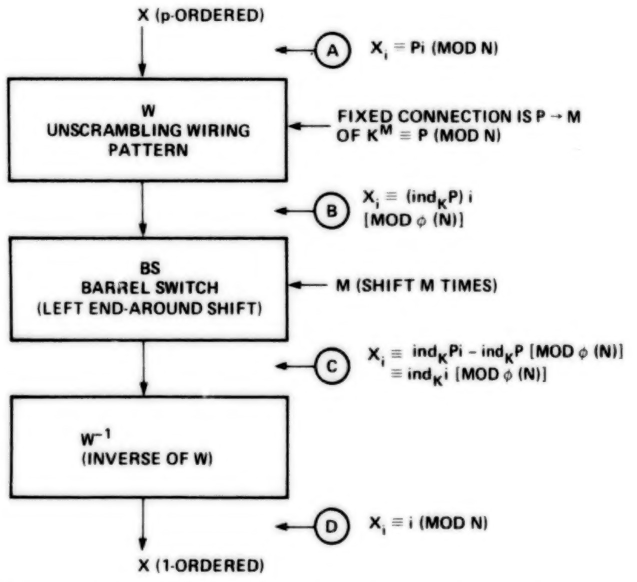
2. In order to obtain $X_i \equiv (\text{ind}_k p)i \pmod{\phi(N)}$ at point-B, W is wired according to the index of p of the unscrambling equation

$$k^m \equiv p \pmod{N}$$

In this equation, $m \equiv \text{ind}_k p$; and thus, the wiring is from p to m for $p = 0,1,2, \ldots, N-1$.

3. At points B and C, the input of $X_i \equiv \text{ind}_k pi \pmod{\phi(N)}$ is shifted m times left end-around by the barrel switch to obtain the key equation

$$X_i \equiv \text{ind}_k i \pmod{\phi(N)}$$
$$\equiv \text{ind}_k pi - \text{ind}_k p \pmod{\phi(N)}$$



X (p-ORDERED)
A
$X_i \equiv Pi$ (MOD N)
W
UNSCRAMBLING WIRING PATTERN
FIXED CONNECTION IS P → M OF $K^M \equiv P$ (MOD N)
B
$X_i \equiv (\text{ind}_K P)\, i$ [MOD ϕ (N)]
BS
BARREL SWITCH
(LEFT END-AROUND SHIFT)
M (SHIFT M TIMES)
C
$X_i \equiv \text{ind}_K Pi - \text{ind}_K P$ [MOD ϕ (N)]
$\equiv \text{ind}_K i$ [MOD ϕ (N)]
W^{-1}
(INVERSE OF W)
D
$X_i \equiv i$ (MOD N)
X (1-ORDERED)

Figure 7.- Barrel switch implementation of Swanson network according to the key equation, equation (5-10).

Note that $m \equiv \text{ind}_k p$ and left shift is minus. It can be shown that

$$m \equiv \text{ind}_k p \pmod{\phi(N)}$$

as follows. From

$$k^m \equiv p \pmod N$$

it follows that

$$m\ \text{ind}_k k \equiv \text{ind}_k p \pmod{\phi(N)}$$

or

$$m \equiv \text{ind}_k p \pmod{\phi(N)}$$

since $\text{ind}_k k \equiv 1$ by property (3) of the index theorem.

4. The output at point-D is obtained by routing

$$X_i \equiv \text{ind}_k i \pmod{\phi(N)}$$

through W-1, which is the inverse of W. This inverse wiring pattern will obtain i from $\text{ind}_k i$, so that the desired output of

$$X_i \equiv i \pmod N$$

is obtained.

To illustrate the above steps further, an example for $N = 11$ and $k = 2$ is shown in figure 8. The wiring pattern W is constructed according to

$$2^m \equiv p \pmod{11}$$

for $p = 1,2, \ldots, 10$. The wiring direction is from p to m because m is the index of p. Also shown in figure 8 is the unscrambling of a p-ordered vector with $p = 6$, the same as in figure 6.

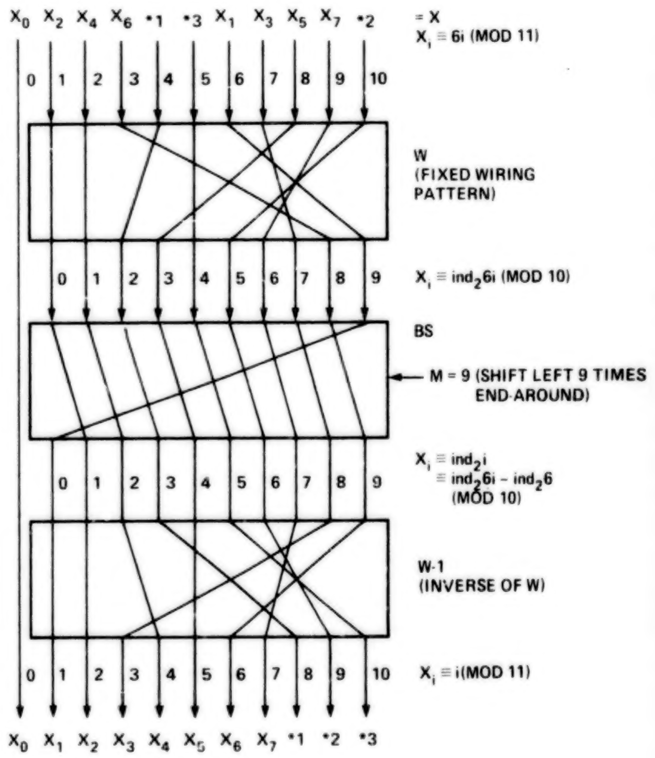


Figure 8.- Illustration of figure 7 for $N = 11$, $k = 2$, and $p = 6$. If offset of X is not zero, X can be preshifted by another barrel switch prior to input of W.

At this point, it should be pointed out that the description of unscrambling p-ordered vectors has assumed that the offset is zero. That is, X_0 of X is always in memory module 0 or in register 0. If the offset is not zero, then a preshift of X is required. This can be accomplished by a barrel switch prior to the input of the TN.

VII. PROGRAMMING AND APPLICATION

The programming of the TN, for either the L-level k-apart networks or the barrel switch implementation, is identical and very simple. It involves only two parameters: the offset and the shift amount m. These two parameters are available from the compiler of the FMP since the compiler is responsible for laying out the three-dimensional data set in the EM modules. The offset of a p-ordered vector X, as described in section II, is a variable and is simply equal to the memory module address of the first element of X. The parameter p is also a variable and is simply equal to the difference of the memory module addresses of X_{i+1} and X_i of X, for $i = 1,2, . . ., n$. Note that the subscript i of X here is changed to "1 to n" because in FORTRAN matrix, the counting starts with 1, not 0. Once p is known, m can be obtained as the index of p according to $k^m \equiv p \pmod N$.

The primary application of the TN is for unscrambling p-ordered vectors that arise naturally from data allocation in memory modules. The secondary application is for data communication between processors through the EM modules, since there are no direct connections between processors of the FMP as shown in figure 1.

In this section, the derivation of the memory module addresses and a brief description of the TN application to unscramble three-dimensional data sets of turbulent flows are presented. The presentation here is for illustrative purposes only. In practice, the actual techniques used may be different. Also, the application of the TN to perform a perfect shuffle is described briefly without proof.

PROGRAMMING

Before proceeding with the memory module address calculation, the definitions of a few notations are important. Let a three-dimensional data set be represented by D(I,J,K), where I,J, and K are the maximum values in the three dimensions. It follows that D(I) and D(I,J) are one-dimensional and two-dimensional data sets, respectively. If the data layout method is row major order, then I, J, and K represent the row, column, and file, respectively. If, however, the data layout method is column major order, then I, J, and K represent the column, row, and file, respectively. As an example, the data set D(I,J,K) in figure 9 is D(3,5,7) for row major order and is D(5,3,7) for column major order. In this paper, only the column major order data layout method is of interest because of FORTRAN compatibility. For this method, the data layout (similar to figure 3) for N = 11 memory modules is shown in figure 10.

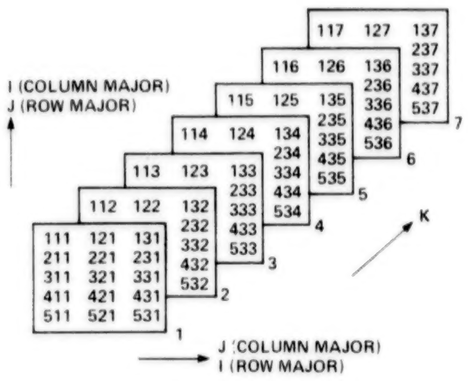


Figure 9.- A three-dimensional data set: for row major, D(I,J,K) = D(3,5,7), for column major, D(I,J,K) = D(5,3,7).

i \ M_j	0	1	2	3	4	5	6	7	8	9	10
0	X	X	X	X	X	X	X	X	111	211	311
1	411	511	121	221	321	421	521	131	231	331	431
2	531	112	212	312	412	512	122	222	322	422	522
3	132	232	332	432	532	113	213	313	413	513	123
4	223	323	423	523	133	233	333	433	533	114	214
5	314	414	514	124	224	324	424	524	134	234	334
6	434	534	115	215	315	415	515	125	225	325	425
7	525	135	235	335	435	535	116	216	316	416	516
8	126	226	326	426	526	136	236	336	436	536	117
9	217	317	417	517	127	227	327	427	527	137	237
10	337	437	537								

Figure 10.- Column major order storage of a three-dimensional data set D(I,J,K) = D(5,3,7).

Let $A_M(i,j,k)$ denote the memory module address of an element in D(I,J,K). It follows that $A_M(i)$ and $A_M(i,j)$ are memory module addresses for D(I) and D(I,J), respectively. In figure 10, the element 111 is the very first element of D(I,J,K), and it is stored in memory module M_8. For this reason, the starting address offset A_0, in this case, is 8. The address offset A_0, sometimes, is also called the base address. If the element 111 is stored in M_0, then $A_0 = 0$. It should be noted that the offset A_0 should not be confused with the offset of a p-ordered vector, which is equal to $A_M(i,j,k)$ of the first element in a vector. In the sequel, the address $A_M(i,j,k)$ will be derived systematically, starting with the one-dimensional data set D(I), and figures 9 and 10 will be used as references.

Let a one-dimensional array, or vector, X have elements X_i for $i = 1,2, . . ., n$. The first element of X, X_1, is stored in a memory module located A_0 memory modules away from M_0. Thereafter, the elements of X are stored in memory modules with succeeding elements at progressively higher numbered modules. In this case, the memory module address of X_i, obviously, is

$$A_M(i) \equiv A_0 + (i - 1) \pmod N \tag{7-1}$$

In a two-dimensional data set D(I,J), the memory module address $A_M(i,j)$ of the element X_{ij} (assuming column major) is

$$\begin{aligned} A_M(i,j) &\equiv A_M(i) + I(j - 1) \pmod N \\ &\equiv A_0 + (i - 1) + I(j - 1) \pmod N \end{aligned} \tag{7-2}$$

The proof of equation (7-2) is by the observation that the term $A_M(i)$ is simply the address of X_i in a column in D(I,J). For the first column, the term $I(j - 1) = 0$ because $j = 1$. Therefore, $A_M(i)$ can also be interpreted as the address of X_i in the first column. After X_i in the first column is exhausted, the address of X_i in the second column is incremented by I, which is the column dimension. In the third column and thereafter, the address of X_i is incremented by 2I, 3I, . . ., I(j - 1), in that order. This concludes the proof.

In a three-dimensional data set $D(I,J,K)$, the memory module address $A_M(i,j,k)$ of the element X_{ijk} (assuming column major) is

$$A_M(i,j,k) \equiv A_M(i,j) + IJ(k-1) \pmod N$$

$$\equiv A_0 + (i-1) + I(j-1) + IJ(k-1) \pmod N \qquad (7\text{-}3)$$

The proof for equation (7-3) is similar to that for equation (7-2). The term $IJ(k-1)$ is the increment for the planes. For the first plane, $IJ(k-1) = 0$ because $k = 1$. Thereafter, the plane is incremented by $IJ, 2IJ, \ldots, IJ(k-1)$. Referring to figure 10, the following examples will illustrate the application of equation (7-3).

<u>Example 7-1</u>. The row vector consists of row 2 in plane 1 and row 2 in plane 2 is (211, 221, 231, 212, 222, 232). In figure 10, the layout of this vector is

0	1	2	3	4	5	6	7	8	9	10
*	232	212	221	*	*	*	222	231	211	*

To fetch this vector, the offset and the p, and hence m, are required. These two parameters can be obtained by calculating $A_M(2,1,1)$ and $A_M(2,2,1)$ as follows:

$$A_M(2,1,1) \equiv 8 + (2-1) + 5(1-1) + 5 \times 3(1-1)$$

$$\equiv 9 \pmod{11}$$

$$A_M(2,2,1) \equiv 8 + (2-1) + 5(2-1) + 5 \times 3(1-1)$$

$$\equiv 3 \pmod{11}.$$

The offset and p are then,

$$\text{offset} = A_M(2,1,1) = 9$$

$$p = |A_M(2,1,1) - A_M(2,2,1)| - 1 = 5$$

The minus 1 for the p calculation is necessary because the memory module numbering starts with 0, not 1.

<u>Example 7-2</u>. The file vector consists of (531, 532, 533, 534, 535, 536, 537) in figure 10 is

0	1	2	3	4	5	6	7	8	9	10
531	534	537	*	532	535	*	*	533	536	*

The offset and p are:

$$\begin{aligned}\text{offset} &= A_M(5,3,1)\\ &\equiv 8 + (5 - 1) + 5(3 - 1) + 5 \times 3(1 - 1)\\ &\equiv 0 \pmod{11}\end{aligned}$$

$$p = |0 - A_M(5,3,2)| - 1 = 5 - 1 = 4$$

The development of the three-dimensional address equation $A_M(i,j,k)$ can be extended to n dimensions. For notational convenience, let the n-dimensional address equation be $A_M(i_1,i_2, . . ., i_n)$. The progression of this development, starting with $n = 4$, is (all equations are modulo N):

$$A_M(i_1,i_2,i_3,i_4) \equiv A_M(i_1,i_2,i_3) + I_1I_2I_3(i_4 - 1)$$

$$A_M(i_1,i_2,i_3,i_4,i_5) \equiv A_M(i_1,i_2,i_3,i_4) + I_1I_2I_3I_4(i_5 - 1)$$

$$\vdots$$

$$A_M(i_1,i_2, . . ., i_n) \equiv A_M(i_1,i_2, . . ., i_{n-1}) + I_1I_2 . . . I_{n-1}(i_n - 1)$$

Once the offset and the p parameters are obtained, the CU obtains m from $k^m \equiv p \pmod N$ in some manner, probably by table lookup. The CU effects this control over the TN and connects the 512 processors to 521 EM modules in some p-order. From this discussion, it should be clear that a processor has absolutely no control over its EM module connection. For this reason, any EM access by a processor must be coordinated with the CU and also wait for the other 511 processors to come to a stop or a synchronization point before such an access can be executed. These wait and synchronization events can cause a degradation in computation performance. There are, however, exceptions to these restrictions. The discussion on these exceptions, of course, is beyond the scope of this paper. After the processor-memory connection is made by the TN, each processor is responsible for generating the address $A(i,j,k)$ to access a location within its connected EM module. This address can be obtained from $A_M(i,j,k)$, as follows:

$$A(i,j,k) = \left\lfloor \frac{|A_M(i,j,k)|}{N} \right\rfloor \tag{7-4}$$

where $|A_M(i,j,k)|$ denotes the true value of $A_M(i,j,k)$ without the congruence. The following examples will illustrate the application of equation (7-4).

<u>Example 7-3</u>. In example 7-1, the element 211 is in M_9. Its location in M_9 is

$$A(2,1,1) = \left\lfloor \frac{9}{11} \right\rfloor = 0$$

The element 211 is in M_3. Its location in M_3 is

$$A(2,2,1) = \left\lfloor \frac{14}{11} \right\rfloor = 1$$

<u>Example 7-4</u>. In example 7-2, the element 531 is in M_0. Its location in M_0 is

$$A(5,3,1) = \left\lfloor \frac{22}{11} \right\rfloor = 2$$

The element 537 is in M_2. Its location in M_2 is

$$A(5,3,7) = \left\lfloor \frac{|A_M(5,3,7)|}{11} \right\rfloor = \left\lfloor \frac{112}{11} \right\rfloor = 10$$

This completes the discussion on TN programming.

APPLICATION

The primary application of the TN, as mentioned earlier, is for unscrambling p-ordered and pq-ordered vectors. It was shown that these vectors arise naturally from the storage allocation of three-dimensional data sets. In practice, these data set sizes for the present Reynolds averaged Navier-Stokes flow codes are typically (100 × 100 × 100) and (200 × 50 × 100). The computational methods to solve these equations are often specially split, which requires that memory accesses must be made in all three directions. In each of the I, J, and K directions, two memory access patterns are required. These two patterns are the row access and the column access. This gives a total of six possible access patterns. Consistent with earlier definitions, let D(I,J,K) denote the access pattern when computing in the K direction with column access. Then, with respect to the six possible permutations of the I, J, and K directional indices, these six access patterns are:

D(I,J,K)	K direction, column access
D(J,I,K)	K direction, row access
D(I,K,J)	J direction, column access
D(K,I,J)	J direction, row access
D(J,K,I)	I direction, column access
D(K,J,I)	I direction, row access

The above six access patterns are also presented in reference 2, appendix A, in a somewhat cryptic manner. In each of these six access patterns, the TN must be programmed to unscramble p-ordered and pq-ordered vectors and also, vectors that have memory access conflicts.

The TN can also be programmed to perform a perfect shuffle (ref. 23). The following description is given without proof. Let X be a 1-ordered vector of $n = 2^{\ell}$ elements, where ℓ is a positive integer. It is assumed that the elements of X are stored in N memory modules with succeeding elements at progressively higher numbered modules. To perform a perfect shuffle on X, the TN can be programmed as follows:

1. Treat X as a p-ordered vector. Unscramble X with $p = n/2$. The resultant vector is a pq-ordered vector.

2. Shift the pq-ordered vector as many times as necessary to form the perfect shuffle.

An example for $n = 2^3$ and N = 11 is shown in figure 11. In this figure, the TN is set to m = 2, corresponding to p = 4. The unscrambled vector is a pq-ordered vector with p = 1 and q = 1. If this pq-ordered vector is rearranged three more times by appropriate left shifting, the result obtained is $(X_0X_4X_1X_5X_2X_6X_3X_7)$, which is a perfect shuffle of eight elements.

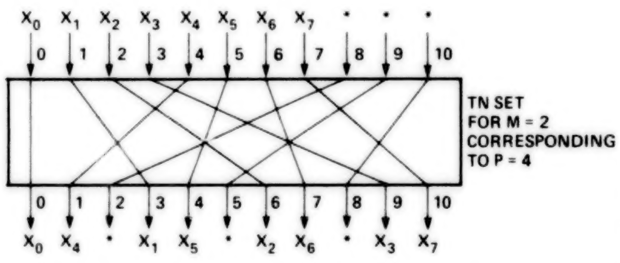

X_0 X_1 X_2 X_3 X_4 X_5 X_6 X_7 • • •
0 1 2 3 4 5 6 7 8 9 10
TN SET FOR M = 2 CORRESPONDING TO P = 4
0 1 2 3 4 5 6 7 8 9 10
X_0 X_4 • X_1 X_5 • X_2 X_6 • X_3 X_7

Figure 11.- TN used to perform a perfect shuffle of 8 elements by first converting X to a pq-ordered vector.

CONCLUSION

A tutorial description of the Burroughs NASF TN has been presented. Basically, the TN is a bidirectional programmable combinational logic network that connects a 521-module EM to an array of 512 processors, where 521 is selected as the smallest prime number greater than 512. The TN is one method of solving the traditional memory-processor connection problem in parallel array processors. The primary application of the TN is for unscrambling p-ordered and pq-ordered vectors. The TN, if programmed appropriately, can also be used to perform a perfect shuffle. The advantage of the TN is simplicity and ease of control. The disadvantage of the TN, in the general sense, is low performance.

It was shown that p-ordered and pq-ordered vectors arise naturally from storage allocation of two-, three-, and n-dimensional data sets in the EM of the FMP, which is similar in architecture to that of the ILLIAC IV array memory system. If the vector is p-ordered, the TN can unscramble it to a 1-ordered vector in one cycle. If the vector is pq-ordered, or some other permutation-ordered, several cycles are required for unscrambling. Unlike other more complicated and powerful permutation networks, the TN cannot, in general, unscramble non-p-ordered vectors in one cycle. This can be a disadvantage.

The programming of the TN is relatively simple when compared to the other permutation networks. This is an advantage. Two programming parameters, the offset and the shift amount, are controlled by the CU. Since the compiler is responsible for laying out the data set in EM, the compiler can furnish the information to the CU for calculating these two parameters. In this context, the CU has absolute control over the TN. For this reason, any EM access by a processor must be coordinated with the CU and also needs to wait for the other 511 processors to come to a stop or a synchronization point before such an access can be executed. These wait and synchronization events can degradate the performance of a computation. This can be a disadvantage.

The design and the implementation of the TN is simple. This is an advantage. The design is based upon the Swanson network (ref. 8). The Swanson network is a k-apart interconnection network constructed according to the theory of cyclic groups and the theory of primitive roots. The barrel switch, it turns out, can be used to implement the Swanson network, and hence the TN. The implementation consists of a single level of barrel switch plus a fixed wiring pattern and its inverse. The result is a very simple network.

In the FMP of the NASF, whether or not the TN should be used to solve the traditional memory-processor connection problem is a matter of complexity versus flexibility. The TN is simple but inflexible. Other permutation networks, such as the Benes network (ref. 4), are somewhat more powerful and more complex. The research and development of these networks are technically challenging and also are important in advancing the art of parallel computing. It is hoped that this effort will be encouraged and continued.

ACKNOWLEDGMENTS

The author thanks the other members of the NASF Project team, consisting of F. R. Bailey, D. L. Fisher, A. W. Hathaway, and K. G. Steven, Jr., for their help and encouragement in writing this paper. He also thanks his team members and colleagues at the Institute for Advanced Computation, consisting of H. Brown, G. F. Feierbach, P. Hiss, and D. K. Stevenson for reading and commenting on the work reported herein.

Ames Research Center
National Aeronautics and Space Administration
Moffett Field, Calif., 94035, October 10, 1978

APPENDIX A

A Method to Compute N^k

In general, the computation of N^k, where both k and N are positive integers, requires k multiplications. The method to be presented, an old and undocumented method in number theory, requires at most

$$2 \log_2 (k + 1) = 2 [\ell n(k + 1)/\ell n 2]$$

multiplications. To compute N^k, express k as an n-bit binary number as follows:

$$k = k_0 2^0 + k_1 2^1 + . . . + k_{n-1} 2^{n-1} \qquad \text{(A-1)}$$

where $k_i = 0$ or 1, $i = 0,1,2, . . ., n - 1$. Next,

$$N^k = N^{\left(k_0 2^0 + k_1 2^1 + ... + k_{n-1} 2^{n-1}\right)}$$

$$= N^{k_0 2^0} N^{k_1 2^1} . . . N^{k_{n-1} 2^{n-1}} \qquad \text{(A-2)}$$

In equation (A-2), N^i can be obtained in n, $n << k$, multiplications plus the preparation of a powers-of-2 table, at most n entries. The value of n can be equated to k by applying the identity

$$2^n - 1 = 2^0 + 2^1 + . . . + 2^{n-1} \qquad \text{(A-3)}$$

to equation (A-1). In equation (A-1), there are at most n terms. The upper bound for k is

$$k = 2^n - 1 \qquad \text{(A-4)}$$

or

$$n = \log_2 (k + 1) \qquad \text{(A-5)}$$

Thus, the number of multiplications required to compute N^k is

$$2n = 2\log_2 (k + 1) \qquad \text{(A-6)}$$

n multiplications for equation (A-2), and n multiplications for preparing the powers-of-2 table. This method can be summarized as follows:

1. Prepare a table by computing 2^{2^i} for $i = 0,1,2, . . .$, up to about $i = k_{n-1}$.

2. Form the product per equation (A-2) for those terms for which $k_i = 1$.

Example 1. Compute 2^{20}

1. $20 = \begin{array}{ccccc} 4 & 3 & 2 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{array}$

i	0	1	2	3	4
2^{2^i}	2	4	16	256	65536

2. $2^{20} = 2^{2^4}\, 2^{2^2} = 65536 \times 16 = 1{,}048{,}576$

Example 2. Show that $2^{260} \equiv 1$ modulo 521

1. $260 = \begin{array}{ccccccccc} 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{array}$

i	0	1	2	3	4	5	6	7	8
2^{2^i}	2	4	16	256	411	117	143	130	228

(mod 521)

2. $2^{260} = 2^{2^8}\, 2^{2^2} = 228 \times 16 = 3648$

$3648 \equiv 1$ modulo 521

REFERENCES

1. Numerical Aerodynamic Simulation Facility, Preliminary Study. (Final Report, Burroughs Corporation, Paoli, PA, NASA-Ames Contract No. NAS2-9456.) NASA CR-152061 - vol. 1; NASA CR-152062 - vol. 2, 1977.

2. Numerical Aerodynamic Simulation Facility, Preliminary Study Extension. (Final Report, Burroughs Corporation, Paoli, PA, NASA-Ames Contract No. NAS2-9456.) NASA CR-152106, 1978.

3. Barnes, George H.; Brown, Richard M.; et al.: The ILLIAC IV Computer. IEEE Trans. on Computers, vol. C-17, Aug. 1968, pp. 746-757.

4. Benes, V. E.: Optimal Rearrangeable Multistage Connecting Networks. Bell System Tech. J., vol. 43, July 1964, pp. 1641-1656.

5. Waksman, Abraham: A Permutation Network. J. Assoc. Comp. Machinery, Jan. 1968, pp. 159-163.

6. Lawrie, Duncan H.: Access and Alignment of Data in an Array Processor. IEEE Trans. on Computers, vol. C-24, Dec. 1975, pp. 1145-1155.

7. Feierbach, Gary; and Stevenson, David: A Feasibility Study of Programmable Switching Networks for Data Routing. Phoenix Project Memorandum No. 003, Revised, May 1977. Institute for Advanced Computation, Sunnyvale, CA.

8. Swanson, Roger C.: Interconnections for Parallel Memories to Unscramble p-Ordered Vectors. IEEE Trans. on Computers, vol. C-23, Nov. 1974, pp. 1105-1115.

9. Budnik, Paul; and Kuck, David J.: The Organization and Use of Parallel Memories. IEEE Trans. on Computers, vol. C-20, Dec. 1971, pp. 1566-1569.

10. Millstein, Robert E.: Control Structures in ILLIAC IV FORTRAN. Comm. Ass. Comput. Mach., vol. 16, Oct. 1973, pp. 621-627.

11. Shapiro, Henry D.: Theoretical Limitations on the Efficient Use of Parallel Memories. IEEE Trans. on Computers, vol. C-27, May 1978, pp. 421-428.

12. Burton, David M.: Elementary Number Theory. Allyn and Bacon, Inc., 1976.

13. Thompson, Robert C.; and Yaqub, Adil: Introduction to Abstract Algebra. Scott, Foresman, and Co., 1970.

14. Booth, Taylor L.: Sequential Machines and Automata Theory. In Fundamentals of Abstract Algebra, Chapter II, John Wiley and Sons, Inc., 1967.

15. Andrews, George E.: Number Theory. Saunders, Inc., 1971.

16. Niven, Ivan; and Zuckerman, Herbert S.: An Introduction to the Theory of Numbers. 3rd Edition, John Wiley and Sons, Inc., 1972.

17. Abramowitz, M.; and Stegun, I. A. (Eds.): Handbook of Mathematical Functions. Nat. Bureau of Standards, App. Math. Series #55, U.S. Gov. Printing Office, Washington, D.C., June 1964.

18. Davis, Robert L.: Uniform Shift Network. Computer, vol. 7, Sept. 1974, pp. 60-71.

19. Burroughs Corporation: Digital Computer Principals. Second ed., McGraw-Hill Book Co., 1969, pp. 253-257.

20. Schlitt, G.: 8243 Eight-Bit Position Scalar. Signetics Corp., Applications Memo 110, Dec. 1970.

21. Davis, R. L.; and McGonagle, J. D.: Advanced Multiprocessor. U.S. Air Force Technical Report, AFAL-TR-69-308, June 1970.

22. Lim, Raymond S.: A Barrel Switch Design. Computer Design, vol. 11, Aug. 1972, pp. 76-79.

23. Stone, Harold S.: Parallel Processing with the Perfect Shuffle. IEEE Trans. on Computers, vol. C-20, Feb. 1971, pp. 153-161.

1. Report No. NASA TP- 1426	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle NASF TRANSPOSITION NETWORK: A COMPUTING NETWORK FOR UNSCRAMBLING p-ORDERED VECTORS		5. Report Date April 1979
		6. Performing Organization Code
7. Author(s) Raymond S. Lim		8. Performing Organization Report No. A-7645
		10. Work Unit No. 366-18-50
9. Performing Organization Name and Address Ames Research Center, NASA Moffett Field, Calif. 94035		11. Contract or Grant No.
		13. Type of Report and Period Covered Technical Paper
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546		14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

This paper presents a tutorial description of a transportation network (TN) proposed by the Burroughs Corporation for the Numerical Aerodynamic Simulation Facility (NASF). The description is presented from the viewpoints of design, programming, and application. The TN is a programmable combinational logic network that connects 521 memory modules to 512 processors, where gcd(521,512) = 1. The primary purpose of the TN is to transpose (or unscramble) p-ordered vectors to 1-ordered vectors in one cycle. For unscrambling pq-ordered vectors, the TN speed is degenerated to several cycles. The TN design, which is evolved from the Swanson network, is based upon the concept of cyclic groups from abstract algebra and primitive roots and indices from number theory. The design can be implemented by one level of barrel switch plus a fixed wiring pattern and its inverse. The connection of this fixed wiring pattern is from p to m according to $k^m \equiv p \pmod{N}$, where k is a primitive root of the prime N, m is the index of p relative to k, and p is an element of the cyclic group of order N - 1 generated by k. The programming of the TN is very simple, requiring only 20 bits: 10 bits for offset control and 10 bits for barrel switch shift control. This simple control is executed by the control unit (CU), not the processors. For this reason, any memory access by a processor must be coordinated with the CU and wait for all other processors to come to a synchronization point. These wait and synchronization events can be a degradation in performance to a computation. The TN application is for multidimensional data manipulation, matrix processing, and data sorting, and can also perform a perfect shuffle. Unlike other more complicated and powerful permutation networks, the TN cannot, if possible at all, unscramble non-p-ordered vectors in one cycle.

17. Key Words (Suggested by Author(s)) Memory-processor connection networks Data alignment networks Parallel computers Parallel computations	18. Distribution Statement Unlimited STAR Category — 62

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 38	22. Price* $4.00

*For sale by the National Technical Information Service, Springfield, Virginia 22161

34

90

50

END

Sept. 14, 1979